Introduction to Object-Oriented Programming (OOP)

CS 3: Computer Programming in Java
Objectives

- Learn the basic terms of object-oriented programming
- Discuss the String class in Java
- Talk about more characteristics of object-oriented programming
- Look into the JOptionPane class for input/output of data
- Discover wrapper classes for primitives
Basic Definitions

- **OBJECT:** an encapsulation of data and a set of operations that access and manipulate the data, an instance of a class
- **CLASS:** a template for creating objects
- **PACKAGE:** collection of classes that have a common purpose
  - The Java API (Application Programming Interface) is composed of many packages
String Class

- We will study this class of objects first because they are so helpful in producing output and gathering input in the program.

- We can look at the documentation for this class by using a browser to open the Oracle Java documentation (https://docs.oracle.com/javase/7/docs/api/) and then searching for the String class on the bottom left corner of the scrolling menu.
  - Your browser will show you the java.lang API Specs for the String Class.
Declaring An Instance of a String Object

- NOTE: All objects are reference variables; they hold the address of the place in memory where the actual data is stored
- Three different ways:
  - Declare the String in one line, then initialize it in the next line
    ```java
    String str1;
    str1 = new String("Hello");
    ```
  - Declare and initialize it in one statement:
    ```java
    String str1 = new String("Hello");
    ```
  - Use shortcut method:
    ```java
    String str1 = "Hello";
    ```
Assigning One String to Another

- **With assignment operator:**
  
  ```java
  String str2;
  str2 = str1;
  // both refer to same string = “Hello”
  ```

- **With instance method:**
  
  ```java
  str2 = str1.replace('H','J'); //str2 references “Jello”
  // each one references separate strings
  ```
Comparing One String to Another

- **NOTE:** Case matters!
- **Test for equality:**
  
  ```java
  if (str1.equals(str2)) { .......}
  
  or
  
  if (str1.compareTo(str2) == 0) { .......}
  ```
- **Test for str1 less than str2:**
  
  ```java
  if (str1.compareTo(str2) < 0) { .......}
  ```
- **Test for str1 greater than str2:**
  
  ```java
  if (str1.compareTo(str2) > 0) { .......}
  ```
Now Onto Generalizing This for Object-Oriented Programming

- Defining a class
- The Constructor method
- Member method
Defining a Class

- It is similar to defining a structure
- The class contains members that are data and members that are methods
- The members are divided into three groups of access categories:
  - PRIVATE: can only be accessed by member functions
  - PUBLIC: can be accessed by any function
  - PROTECTED: discussed later with inheritance
- Usually, the data within a class is defined to be private and the methods are defined to be public
Defining a Class (2)

- Variables and method naming conventions
  - Use lowercase
  - If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name

- Examples
  - Variables: radius, area
  - Method: computeArea
The Constructor Method

- A special method that is a part of every class definition is the constructor method.
- The constructor method has the same name as the class name.
- Its purpose is to initialize an object’s data members and perform any other steps that need to be taken at the “birth” of the object.
- It is called when the new operator is used to instantiate an object during its definition.
- There can be more than one constructor method for a class, so that objects can be “born” in different ways.
Member Method

- **Member Method**

- A member method is defined by the template:
  
  modifier(s) return-type methodname(arg list)
  
  {
   declarations.......;
   statements.......;
   return returnval;
  }
Member Method (2)

- Modifiers are public or private and/or static
- If static is used the method is called a class method; otherwise, it is called an instance method
- Return types are void or some other data or object type
- arg list: defines any arguments that may be passed to the method
- Returnval: must match return type
  - Return statement not used if method has void modifier
Example of a Class: ChemicalElement

```java
public class ChemicalElement {
    private String nameOfElement;
    private String chemicalSymbol;
    private int atomicNumber;
    private static int numOfElements=0;
}```
Example of a Class: ChemicalElement (2)

```java
/**
 * The ChemicalElement class enables an object that represents a chemical element from the periodic table of elements.
 * @param name is the name of the element.
 * @param symbol is the chemical symbol of the element.
 * @param number is the atomic number of the element.
 */
public ChemicalElement(String name, String symbol, int number) {
    nameOfElement = name;
    chemicalSymbol = symbol;
    atomicNumber = number;
    numOfElements++;
}
```
Example of a Class: ChemicalElement (3)

```java
/**
 * @return The name of the element.
 */
public String getName()
{
    return nameOfElement;
}

/**
 * @return The chemical symbol of the element.
 */
public String getSymbol()
{
    return chemicalSymbol;
}
```
Example of a Class: ChemicalElement

/**
 * @return the atomic number of the element.
 */
public int getNumber()
{
    return atomicNumber;
}

/**
 * @return the number of elements constructed.
 */
public static int getNumOfElements()
{
    return numOfElements;
}
About Comments from the ChemicalElement Class

- The comments that begin with /** and contain lines that begin with @ that describe the member data, and member methods are used by NetBeans, Eclipse, and other Java IDEs to generate the HTML documentation for your classes.
Using the JOptionPane Class for Input/Output of Data

- Java has a package called javax.swing, which contains a class called JOptionPane
- When the static method showInputDialog of class JOptionPane is called, the program creates a graphical dialog box in which user can enter data for the program
- The data is returned as a String object
- The method showMessageDialog outputs text within a dialog box and showConfirmDialog returns a Boolean choice taken from the user
import javax.swing.JOptionPane;

public class MyElements
{
    // Input data for a ChemicalElement object and display the object.
    // Uses multiple dialog boxes for user interaction.
    public static void main (String[] args)
    {
        String name, symbol, numStr, message;
        int number, again;
        ...
do
{
    name = JOptionPane.showInputDialog("Enter the Element name: ");
    symbol = JOptionPane.showInputDialog("Enter the Element symbol: ");
    numStr = JOptionPane.showInputDialog("Enter the Element number: ");
    number = Integer.parseInt(numStr);
    ChemicalElement e = new ChemicalElement(name,symbol,number);

    message = "The element name is "+e.getName()\n"Its atomic symbol is "+e.getSymbol()\n"Its atomic number is "+e.getNumber();
    JOptionPane.showMessageDialog(null, message);
}
again = JOptionPane.showConfirmDialog (null, "Do Another?");
}
while (again == JOptionPane.YES_OPTION);

message = "The number of elements processed was " +
    ChemicalElement.getNumOfElements();

JOptionPane.showMessageDialog (null, message);
}
Wrapper Classes for Primitives

- A simple definition of wrapper classes is as follows: The Java wrapper classes are used to convert a primitive data type into a Java object.
- This will allow a primitive to be used in and with methods where only objects are expected.
Wrapper Classes for Primitives (2)

<table>
<thead>
<tr>
<th>Java Primitive Data Type</th>
<th>Corresponding Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>void</td>
<td>Void</td>
</tr>
</tbody>
</table>
Wrapper classes also have instance methods in them to parse a String object to its corresponding primitive numerical data type.

When data is input through the JOptionPane DialogBox, it is returned as a String.

In order to establish a value for the primitive int atomicNumber in the ChemicalElement class, we must input the string data, then parse (convert) the string data to a type int.
Wrapper Classes for Primitives (3)

- This is done in the main above with the method parseInt from the Integer wrapper class in the statement:
  
  ```java
  number = Integer.parseInt(numStr);
  ```

- You should look at the javadocs for JOptionPane, and for the Wrapper Classes to get more info about the methods you can use with them
Summary

- Basic terms of object-oriented programming in Java
  - Object
  - Class
  - Package

- Discussed the following for the String class in Java
  - Declaring an instance of a String object
  - Assigning one String to another
  - Comparing one String to another
Summary (2)

- Talked about more characteristics of object-oriented programming
  - Defining a class
    - Public
    - Private
    - Protected
  - The Constructor method
  - Member methods
Summary (3)

- Looked into the JOptionPane class for input/output of data
- Discover wrapper classes for primitives
  - The Java wrapper classes are used to convert a primitive data type into a Java object, which will allow a primitive to be used in and with methods where only objects are expected